Composers' Desktop Project

CDP 'Wavecycle' DISTORT Functions

(with Command Line Usage)

Functions to DISTORT soundfiles via 'pseudo-wavecycles'

DISTORT **AVERAGE**
: Average the waveshape over *N* 'wavecycles'

CLIP
: Clip a signal

DISTORT **CYCLECNT**
: Count 'wavecycles' in soundfile

DISTORT **DELETE**
: Time-contract soundfile by deleting 'wavecycles'

DISTCUT
: Cut sound into elements with falling envelope

DISTMARK
: Interpolate between waveset-groups at marked points

DISTMORE BRIGHT
: Reorder sound segments in order of average zero-crossing rate

DISTMORE DOUBLE
: Double (quadruple etc.) frequency of each waveset

DISTMORE SEGSBKWD
: Reverse certain (sets of) segments

DISTMORE SEGZIG
: Zigzag across tail segments or across whole soundfile

DISTORTT
: Repeat wavesets within given duration

DISTREP
: Timestretch soundfile by repeating wavesets

DISTSHIFT
: Time-shift or swap wavecycles

DISTWARP
: Warp wavecycles by a multiplier

DISTORT **DIVIDE**
: Distortion by dividing 'wavecycle' frequency

DISTORT **ENVEL**
: Impose envelope over each group of *cyclecnt* 'wavecycles'

DISTORT **FILTER**
: Time-contract a sound by filtering out 'wavecycles'

DISTORT **FRACTAL**
: Superimpose miniature copies of source 'wavecycles' onto themselves

FRACTAL WAVE
: Fractally distort an input sound or wavecyle

DISTORT **HARMONIC**
: Harmonic distortion by superimposing 'harmonics' onto 'wavecycles'

DISTORT **INTERACT**
: Time-domain interaction of two sounds

DISTORT **INTERPOLATE**
: Time-stretch file by repeating 'wavecycles' and interpolating between them

DISTORT **MULTIPLY**
: Distortion by multiplying 'wavecycle' frequency

DISTORT **OMIT**
Omit *A* out of every *B* 'wavecycles', replacing them with silence

DISTORT **OVERLOAD**
Clip the signal with noise or a (possibly timevarying) waveform

DISTORT **PITCH**
Pitchwarp 'wavecycles' of sound

DISTORT **PULSED**
Impose regular pulsations on a sound

QUIRK
Distort signal by raising sample values to a power

DISTORT **REFORM**
Modify the shape of 'wavecycles'

DISTORT **REPEAT**
Timestretch soundfile by repeating 'wavecycles'

DISTORT **REPEAT2**
Repeat 'wavecycles' without time-stretching

DISTORT **REPLACE**
The strongest 'wavecycle' in a *cyclecnt* group replaces the others

DISTORT **REPLIM**
Timestretch by repeating 'wavecycles' (below a specified frequency)

DISTORT **REVERSE**
Cycle-reversal distortion in which the 'wavecycles' are reversed in groups

SCRAMBLE
Scramble waveset order randomly or by size and level

DISTORT **SHUFFLE**
Distortion by shuffling 'wavecycles'

SPLINTER
Create splinters by repeating & shrinking selected waveset-group

DISTORT **TELESCOPE**
Time-contract sound by telescoping *N* wavecycles into 1

SEE ALSO:

PARTITION
Partition a mono soundfile into disjunct files in blocks defined by groups of wavesets

Technical Discussion
Description of a 'Pseudo-wavecycle'

The Wavecycle DISTORT Functions
Richard Dobson's Introduction to the T Wishart Wavecycle Distort Functions

The aural results of the DISTORT and other WAVESET processes are highly variable, so the Musical Applications sections below are notably sparse. You'll get the best results by experimenting yourself!

DISTORT AVERAGE – Average the waveshape over *N* 'wavecycles'

Usage

distort average *infile outfile cyclecnt* **-m***maxwavelen* **-s***skipcycles*

Parameters

infile – soundfile to process (mono only)
outfile – output soundfile
cyclecnt – number of cycles over which to average (Range: > 1)

cyclecnt may vary over time.

-m*maxwavelen* – maximum permissible wavelength in seconds (Default: 0.50)
-s*skipcycles* – (integer) number of wavecycles to skip at start of file

Understanding the DISTORT AVERAGE Process

DISTORT AVERAGE performs a mathematical averaging of the data in *cyclecnt* pseudo-wavecycles. The effect is more akin to a loss of resolution than the blurring which might be expected. Values below 10 retain some semblance of the original, while values of, for example, 100 seem to create a kind of 'sample hold' effect. For modest distortion, values 5 or less are recommended.

Musical Applications

Mushy, watery, aspects of tracing, discrete sample-hold effects...

End of DISTORT AVERAGE

CLIP – Clip a signal

Usage

clip clip 1 *infile outfile level*
clip clip 2 *infile outfile fraction*

Modes

1 Clip signal at specified level
2 Clip half-waveforms at specified fraction

Parameters

infile – input soundfile
outfile – output soundfile
level – level in original signal at which to clip (Range: 0 to 1)
An error message is given if none of the original signal exceeds *level*.
fraction – fraction of original signal at which to clip (Range: 0 to 1; 1= no change)

Understanding the CLIP Process

Clipping a signal cuts off the top of the waveform when the level exceeds 100%, introducing distortion. **CLIP** produces this distortion at lower signal levels.

In Mode 1, *level* sets the level at which to clip the signal. This acts as a gate value: if too high, an error message may remind the user that the signal never reaches that level. If set lower, the signal is raised (or appears to be raised) so that all values that exceed that level are clipped. (In many cases, this may not result in an obvious distortion, because most of the signal may be unclipped.)

In Mode 2, half-waveforms are clipped at a given *fraction* of the original. This seems to give a more obvious distortion, even though the signal level appears to change little. Again, a lower value produces more clipping and more distortion.

Musical Applications

...

End of CLIP

DISTORT CYCLECNT – Count 'wavecycles' in soundfile

Usage

distort cyclecnt *infile*

Parameters

infile – soundfile to examine (mono only)

Understanding the DISTORT CYCLECNT Function

DISTORT CYCLECNT checks the waveform of the *infile* for zero crossings, determining how many segments lie between these crossings. It then displays this figure on screen.

Musical Applications

This information – the number of 'waveycles' in a file – can help predict the level of distortion a given process might produce.

Knowing the total number of 'wavecycles' also enables you to set a suitable value for several parameters found in the DISTORT programs.

End of DISTORT CYCLECNT

DISTORT DELETE –Time-contract soundfile by deleting 'wavecycles'

Usage

distort delete mode *infile outfile cyclecnt* **-s***skipcycles*

Modes

1 One 'wavecycle' in every *cyclecnt* 'wavecycles' is retained
2 The strongest (single) 'wavecycle' in every *cyclecnt* 'wavecycles' is retained
3 The weakest (single) 'wavecycle' in every *cyclecnt* 'wavecycles' is deleted

Parameters

infile – soundfile to process (mono only)
outfile – output, time-contracted, soundfile
cyclecnt – groups of 'wavecycles': really the level of resolution at which the process will work

cyclecnt may vary over time.

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT DELETE Process

Mode **1** dramatically removes data from the *infile*, leaving very little behind. Mode **2**, because it sets out to retain the strongest (i.e., highest amplitude) 'wavecycle' in each set, retains more recognisable features from the original. Mode **3** takes this further be deleting the weakest (i.e., lowest amplitude) 'wavecycle' in each set. The three Modes, therefore, enable the user to target levels of recognisability.

Because the 'wavecycles' are of irregular lengths, the idea of 'resolution' is only relative. However, it helps picture the degree to which the *infile* is divided up into units. Then one 'wavecycle' from each of these units is retained or deleted according to the operation of each mode.

The *skipcycles* parameter makes it possible to have this process begin some time after the start of the sound, e.g., so that its start transient, so vital to the recognition of the source of a sound, is not affected.

Musical Applications

This process achieves a time-compression and textural roughening of the source.

End of DISTORT DELETE

DISTCUT – Cut sound into elements with falling envelope

Usage

distcut distcut 1 *infile generic_outfilename cyclecnt exp* [**-c***limit*]
OR
distcut distcut 2 infile generic_outfilename cyclecnt cyclestep exp [**-c***limit*]

Modes

1 Waveset-groups are adjacent
2 Set gap between waveset-groups

Parameters

infile – input soundfile (**MONO**)
outfile – output soundfile
cyclecnt – number of wavesets in each outfile (Range: 1 to 1000, up to the number in the infile.)
cyclestep (Mode 2) – number of wavesets steps from start of one group to start of next. (Range: 1 to 1000, up to the number in the infile.)
(In mode 1 : cyclestep = cyclecnt: waveset-groups are abutted and disjunct)
exp – envelope decay shape: 1 = linear, >1 = more rapid decay, <1 = less rapid decay
-c*limit* – minimum level of output events to accept (in dB, e.g. 70 = -70dB).

Understanding the DISTCUT Process

DISTCUT cuts segments from the infile, producing multiple outputs, each with a falling envelope. Segment lengths are set by groups of wavesets (*cyclecnt*), defined by zero-crossings. The decay creates enveloped segments that do not end abruptly.

In Mode 1, segments are consecutive, wherease in Mode 2 they are separated by *cyclestep*, which is the number of cycles from the start of one segment to the start of the next. The optional parameter *limit* sets a gate level above which the segment must rise to be accepted.

DISTCUT is is closely related to ENVCUT, which extracts segments to a given length, also with a falling envelope.

Musical Applications

DISTCUT may have uses in creating segments for TEXTURE, or for other creative mixes derived from the same material.

End of DISTCUT

DISTMARK – Interpolate between waveset-groups at marked points

Usage

distmark distmark 1 infile outfile marklist unitlen [**-s**tstretch] [**-r**rand] [**-f**] [**-t**]
distmark distmark 2 infile outfile marklist unitlen [**-s**tstretch] [**-r**rand] [**-f**]

Modes

1 Interpolate between waveset-groups at marked points
2 Interpolate within alternate marked blocks

Parameters

infile – input soundfile (**MONO**)
outfile – output soundfile (**MONO**)
marklist – list of times within source at which to find waveset-groups
unitlen – approximate size of waveset group to find (mS); can vary over time. (Range: 0.5 to 1000 mS)

Minimum *unitlen* < ½ of minimum step between times in *marklist*

-ststretch – timestretch distances between marks, in making output. (Range: 1 to 256)
-rrand – randomise duration of interpolated wavesets (Range: 0 to 1)

Randomisation decreases waveset lengths (heard 'pitch' is higher).

-f – Flip phase of alternate wavesets
-t – Add original (remaining) tail of source sound to output

Understanding the DISTMARK Process

DISTMARK interpolates between wavesets at marked times (*marklist* textfile). The aural effect of this is a stream of short repeated segments, with one block merging into the next block. Mode 2 interpolates within alternate marked blocks.

Unitlen sets the approximate size of the waveset group to find and hence the speed of repetition. Setting this too high may produce the CDP Error "Invalid Data". This is caused by infringing the rule that *unitlen* must be < ½ the minimum step between times in the *marklist*. If *unitlength* is very short (e.g. 20mS), the effect is more like synthesis, or LOOP with a short progression time, and typically with a rather buzzy quality.

Tstretch stretches the overall output length of the process.

Musical Applications

...

End of DISTMARK

DISTMORE – A group of additional waveset distortion programs

Sub-Group of 4:
DISTMORE BRIGHT   **DISTMORE DOUBLE**   **DISTMORE SEGSBKWD**   **DISTMORE SEGZIG**

Usage

distmore NAME (mode) *infile outfile (parameters)*
where NAME can be any one of: **bright double segsbkwd segszig**
Type **distmore bright** for the Usage of **distmore bright** .. etc.

DISTMORE BRIGHT – Reorder sound segments in order of average zero-crossing rate

Usage

distmore bright 1-3 infile outfile marklist [**-s***splicelen* **-d**]

Modes

Extract data ...
1 From Heads, & from Tails cut to segments, size approx equal to Heads
2 From Heads and Tails, as defined by *marklist*
3 From Tails only

Parameters

infile – input mono soundfile
outfile – output mono soundfile
marklist – a list of timemarks in source, marking (paired) Heads and Tails, e.g.: consonant onset, and vowel continuation of source. (It is assumed that the first mark is at a Head segment.)
-s*splicelen* – length of splice. (Range: 2 to 15 mS)
-d – output in decreasing order of brightness. (Default: increasing)

Understanding the DISTMORE BRIGHT Process

DISTMORE BRIGHT is one of many processes using wavesets – pseudo-cycles based on zero-crossings. In this process, it re-orders elements based on the average zero-crossing rate.

Segments are selected from a datafile list of times (*marklist*). These are paired into what are called 'Heads' and 'Tails': a Head is taken to mark a 'consonant' onset in the source and a Tail its 'vowel' continuation. The first mark is assumed to start a Head segment. At least two pairs of time-values must be given.

Note that the **-d** flag produces a different order of segments.

Musical Applications

The process can be an effective way of re-ordering the elements in a sound. It is useful for the list of times to match the onset of events in the source soundfile. For vocal sounds, this would normally be the start of each syllable; in a melodic sequence, the start of each note; for drumming, the start of each beat or drumstroke, etc. The program then successfully re-orders the segments, splicing them together.

End of DISTMORE BRIGHT

DISTMORE DOUBLE – Double (quadruple etc.) frequency of each waveset

Usage

distmore double *infile outfile mult*

Parameters

infile – input soundfile (MONO)
outfile – output soundfile (MONO)
mult – octave step up (Range: 1 to 4 octaves, possibly fractional)

Understanding the DISTMORE DOUBLE Process

DISTMORE DOUBLE is a variant of DISTORT MULTIPLY. Like the latter program, wavesets (pseudo-wavecycles) are raised in frequency. However, depending on the material, the perceived pitch may remain the same, but with an altered brightness.

Musical Applications

...

End of DISTMORE DOUBLE

DISTMORE SEGSBKWD – Reverse certain (sets of) segments

Usage

distmore segsbkwd 1-9 *infile outfile marklist*

Modes

1 Reverse Tails
2 Reverse Heads
3 Reverse Head+Tail pairs
4 Reverse Head & Tail+Head+Tail set
5 Reverse Head+Tail+Head+Tail set
6 Reverse Head & Tail+Head+Tail+Head+Tail set
7 Reverse Head+Tail+Head+Tail+Head+Tail set
8 Reverse Head & Tail+Head+Tail+Head+Tail+Head+Tail set
9 Reverse Head+Tail+Head+Tail+Head+Tail+Head+Tail set

Parameters

infile – input soundfile (MONO)
outfile – output soundfile (MONO)
marklist – a list of timemarks in source, marking (paired) Heads and Tails, e.g., consonant onset, and vowel continuation of source. It is assumed that the first mark is at a Head segment.

Understanding the DISTMORE SEGSBKWD Process

The process reverses selected segments given in a datafile list of times (*marklist*). The times are paired into what are called Heads and Tails: a Head typically marks a consonant onset in the source and a Tail its vowel continuation. It is assumed that the first mark is at a Head segment. At least two pairs of time-values must be given. The various modes offer a wide range of Head and Tails combinations.

As with DISTMORE BRIGHT, the Head and Tails principle can be applied to different types of material, such as vocal syllables, notes in a melodic sequence, or drum beats.

A similar process **DISTORT REVERSE** also reverses groups of wavesets, but using a given number of wavecycles. DISTMORE SEGSBKWD produces more coherent segments by identifying them by times.

Musical Applications

DISTMORE SEGSBKWD is a reasonably effective way of reversing the elements in a sound, without reversing their order. If followed by *modify radical 1* (REVERSE), the elements are played forwards, as in the source, but the order of elements is reversed.

End of DISTMORE SEGSBKWD

DISTMORE SEGZIG – Zigzag across tail segments or across whole soundfile

Usage

distmore segszig 1 *infile outfile marklist repets* [**-s***shrinkto*] [**-p***prop*] [**-l**]
distmore segszig 2 *infile outfile repeats* [**-s***shrinkto*] [**-p***prop*] [**-l**]
distmore segszig 3 *infile outfile repeats dur* [**-s***shrinkto*] [**-p**prop]

Modes

1 Zigzag across tail segments of a soundfile while playing it
2 Zigzag across entire soundfile
3 Zigzag across entire soundfile, specifying proportion of tail to use

Parameters

infile – input soundfile
outfile – output soundfile
marklist (Mode 1) – a list of times in source, marking (paired) Heads and Tails, e.g. consonant onsets, and vowel continuations in source. (The first mark is assumed to be a Head segment). In Modes 2 and 3, the whole file is processed.
repeats – number of zigzags. (Range: 1 to 64; can vary over time)
-s*shrinkto* – if set to zero, has no effect. Otherwise Zigzags contract to minimum size *minsiz* in mS (Range 31 upwards).
dur (Mode 3) – duration of zig-zagged output (secs). (Range: from *repeats*+1 x File-length to 64 x infile-length)
-p*prop* – proportion of Tail to use. Default: all of it. (Can vary over time)

> CARE: If the length of used-portion of any particular tail is too short, (less than *shrinkto* size), zigs for that tail will not shrink.

-l – shrink zigs logarithmically (Default: linear shrink)

Understanding the DISTMORE SEGSZIG Process

DISTMORE SEGSZIG applies the idea of zig-zagging within a soundfile to waveset groups. Zig-zagging (see esp. EXTEND ZIGZAG) involves playing a segment forwards and then backwards. Like other processes within the DISTMORE program (BRIGHT and SEGSBKWD), this function can use (in Mode 1) a list of times that are assumed to represent paired Heads and Tails of vocal sounds: a Head typically marks a consonant onset in the source and a Tail its vowel continuation. At least two pairs of time-values must be given and the first marker is taken to be a Head segment. *Repeats* sets the number of repeats required (including the reversal). *Dur* (Mode 3 only) sets the outfile-length. The optional parameter *shrinkto* allows the zigzags to shrink to a minimum size, while *prop* optionally allows less than 100% of the tail to be used.

Musical Applications

As with DISTMORE BRIGHT and SEGSBKWD, the Head and Tails principle can be applied to different types of material, such as vocal syllables, notes in a melodic sequence, or drum beats. It is useful to give the start times of these elements for Mode 1, but similar segments are extracted in an "intelligent" way in Modes 2 and 3.

End of DISTMORE SEGSZIG

DISTORTT – Repeat wavesets within given duration

Usage

distortt repeat *infile outfile gpcnt rpt offset dur* [-*t*]

Parameters

infile – input soundfile (MONO)
outfile – output soundfile (MONO)
gpcnt – number of wavesets in the group to be repeated.
rpt – number of repetitions of each waveset group.
offset – time to skip before starting waveset process: sound in offset is prefixed before the processed sound.
dur – required duration of output.
-t – telescope: skip wavesets so output is of similar size to input.

Understanding the DISTORTT Process

DISTORTT is a variant of DISTORT REPEAT and DISTREP. In this case, the outfile length is specified directly, rather than as a number of wavesets. However, setting the **-t** flag produces an outfile of similar length to the infile, by skipping some wavesets.

The *offset* parameter specifies how much of the source sound to keep before starting the process.

Musical Applications

End of DISTORTT

DISTREP – Timestretch soundfile by repeating wavesets

Usage

distrep distrep *mode infile outfile multiplier ccyleccnt* [**-k***skipcycles*] [**-s***splicelen*]

Modes

1 Timestretch file by repeating 'wavecycles'
2 Repeat 'wavecycles', but skip cycles to avoid timestretch

Parameters

infile – input soundfile (MONO)
outfile – output soundfile (MONO)
multiplier – no. of times each wavecycle (group) repeats. (Range: 2 to 32767, integer)
cyclecnt – (integer) is the number of wavecycles in repeated groups. (Range: 1 to 32767)
-k*skipcycles* – number of cycles to ignore (skip over) at start of file. (Range: 0-32767)
-s*splicelen* – mS splices of repeated blocks (Range: 0-50 mS; default 15 mS)

multiplier and *cyclecnt* may vary over time

Understanding the DISTREP Process

DISTREP is an updated version of DISTORT REPEAT, with the option of added splices between repeated blocks.

The repetition of the 'wavecycles' stretches out the sound, making it both longer and more granular in texture. This granularity is increased if (increasingly larger) groups of *cyclecnt* 'wavecycles' are used: then the whole group repeats *multiplier* times.

Musical Applications

Like DISTORT REPEAT, DISTREP produces long, grainy (distorted) sounds. The sense of stretching out the original is very apparent.

As noted with DISTORT REPEAT, using a small group of cycles (e.g. 10 or less) and a large-ish number of repetitions (e.g. 10-20), arbitrary melodies can arise. These could then be quantised to a given scale with REPITCH QUANTISE.

End of DISTREP

DISTSHIFT – Time-shift or swap wavecycles

Usage

distshift distshift 1 *infile outfile grpcnt shift*
distshift distshift 2 *infile outfile grpcnt*

Modes

1 Shift alternate (groups of) half-wavecycles forward in time
(wrapping back around zero-time if pushed beyond sound end)
2 Swap alternate half-wavecycle (groups)

Parameters

infile – input soundfile (MONO)
outfile – output soundfile (MONO)
grpcnt – size of elements to operate on:

1 = single half-waveset
2 = 1 waveset + single half-waveset
3 = 2 wavesets + single half-waveset
etc.
(Range: 1 to 32767)

shift – move alternate groups forward by *shift* waveset(group)s. (Range: 1 to 32767)

Understanding the DISTSHIFT Process

DISTSHIFT shifts alternate wavesets (pseudo-wavecyles) in time or swaps them. The time-shift is expressed as a number of half-cycles, which can vary from source to source. Large group sizes create fairly random and radical exchanges of material.

Musical Applications

...

End of DISTSHIFT

DISTWARP – Warp wavecycles by a multiplier

Usage

distwarp distwarp 1 *infile outfile warp*
distwarp distwarp 2 *infile outfile warp wavesetcnt*

Modes

1 Warp-multiplier increments samplewise
2 Warp-multiplier increments wavesetwise.

Parameters

warp – (progressive) sample multiplier (Range: 0.0001 to 0.1).
wavesetcnt – the number of wavesets after which the warp value increments (Range: 1 to 100)

Understanding the DISTWARP Process

DISTWARP warps by applying a multiplier, either to each sample or to groups of wavesets. It appears to emphasise amplitude (to be confirmed). In Mode 1 (samplewise), it can produce a distinct pitch - especially towards the top of the *warp* range – this can be varied over time. In waveset mode, the groups are accented and distorted.

Musical Applications

...

End of DISTWARP

DISTORT DIVIDE – Distortion by dividing 'wavecycle' frequency

Usage

distort divide *infile outfile N* [**-i**]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
N – divider (Range: integer only, 2 to 16)
-i – use waveform interpolation: slower but cleaner

Understanding the DISTORT DIVIDE Process

Without altering duration, this process effectively lowers the sound while adding a rough texture.

Musical Applications

This is a useful form of distortion because it roughens the sound without being too violent about it.

End of DISTORT DIVIDE

DISTORT ENVEL – Impose envelope over each group of *cyclecnt* 'wavecycles'

Usage

distort envel 1-2 *infile outfile cyclecnt* **-t***troughing* **-e***exponent*
distort envel 3 *infile outfile cyclecnt troughing* **-e***exponent*
distort envel 4 *infile outfile envfile cyclecnt*

Modes

1 Rising envelope
2 Falling envelope
3 Troughed envelope
4 User-defined envelope

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
envfile – defines user envelope as *time value* pairs (Range of *value* is 0 to 1)

The time units in *envfile* are in fact arbitrary because in each case the envelope is stretched to fit the duration of each *cyclecnt* set.

cyclecnt – number of 'wavecycles' under a single envelope
[**-t**]*troughing* – the trough depth of the envelope (Range: 0 [most troughed] to 1 [least troughed], Default = 0)
-e*exponent* – exponent to shape envelope rise or decay

- < 1 will produce a curve which starts quickly and slows down
- > 1 will produce a curve which starts slowly and speeds up
- Omitting this parameter will result in a linear rise or decay

cyclecnt, *troughing* and *exponent* may vary over time.

Understanding the DISTORT ENVEL Process

The process takes the amplitude envelope data for each group of *cyclecnt* 'wavecycles' and adjusts this data to form a single envelope shape (for that group) according to the mode selected.

Musical Applications

The same pattern repeats (irregularly), being reasonably fine or quite coarse depending on the size of *cyclecnt*. This can be likened to creating a somewhat irregular sawtooth edge on the sound.

The *exponent* and *envfile* parameters can be used to customise/intensify the effect of the enveloping.

Here are some possible results with different values for *cyclecnt* :

- **Small values** (1 – 3), can produce a timbral distortion of the source.
- **Medium values** can produce a granular modification.
- **Very Large values** (64+) can result in the superposition of a repeating envelope (e.g., like a tremolando). But, with a natural rather than a synthetic source, this is likely to have a natural-sounding variability, due to the variable durations of the wavecycles which are being counted.
- It is interesting to **gradually increase** the *cyclecnt* and hear the sound cross these different perceptual boundaries.

One of the most telling applications of DISTORT ENVEL is to create irregular phasing effects. The most important tool for this is a time-varying breakpoint file for *cyclecnt*, which can be used to introduce large variations in the length of each envelope shape. If this is done in two different ways (i.e., two different breakpoint files), then the two resultant (mono) files can be combined with SUBMIX INTERLEAVE to form a stereo file with phased *cyclecnt* envelope shapes.

End of DISTORT ENVEL

DISTORT FILTER – Time-contract a sound by filtering out 'wavecycles'

Usage

distort filter 1-2 *infile outfile freq* [**-s***skipcycles]*
distort filter 3 *infile outfile freq1 freq2* [**-s***skipcycles*]

Modes

- **1** Omit cycles below *freq*
- **2** Omit cycles above *freq*
- **3** Omit cycles below *freq1* and above *freq2*

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
freq – frequency in Hz (Range: 10.0 to 22050.0)
freq1 – frequency in Hz to delete below
freq2 – frequency in Hz to delete above

- *freq*, *freq1* and *freq2* may vary over time.
- **NB:** Time-varying *freq1* and *freq2* may not cross each other, nor be equal.
- *Freq* here relates simply to the length of a 'wavecycle'. A high value signifies a short 'wavecycle', and *v.vs.*

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT FILTER Process

Period and frequency are inverse functions. Therefore it is possible to relate the length of a 'wavecycle' to the frequency it would have were it to recur regularly. This program therefore filters by removing 'wavecycles' shorter or longer than those relating to a specific, user-defined, frequency.

The duration of the *outfile* is affected by this process: because 'wavecycles' are being removed, the *outfile* will be shorter, by varying degrees.

Musical Applications

The aural effect of the DISTORT FILTER process is actually like gating. In gating, you can imagine a horizontal line drawn through the time/amplitude display of a soundfile. If the peaks above the line are retained (by filtering out lower frequencies), you just hear them (joined up). Similarly, if you filter out above the line, the peaks are gone, leaving the lower sound material (joined up).

The difference here is that the process is operating on 'pseudo-wavecycles' according to their length, so here the results are more unpredictable and also distort the sound to some degree, depending on where the horizontal line – the frequency variable(s) – is drawn. Technically, the process is akin to low-, high- and band-pass filters, but aurally it is more like gating.

You can therefore use this procedure to cut out some and distort other material in a sound.

End of DISTORT FILTER

DISTORT FRACTAL – Superimpose miniature copies of source 'wavecycles' onto themselves

Usage

distort fractal *infile outfile scaling loudness* [**-p***pre_attenuation*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
scaling – (integer) division of scale of source wave (Range: 2 to sample_rate/2)
loudness – loudness of scaled component relative to source (Loudness of source is reckoned to be 1.0)

> *scaling* and *loudness* may vary over time.

-p*pre_attenuation* – apply attenuation to *infile* before processing

Understanding the DISTORT FRACTAL Process

Note the very wide range of *scaling*. Because it is a divisor, the larger the value of *scale* the shorter will be the miniature copies to be superimposed.

These superimposed copies can be made to increase (be careful!) or decrease in amplitude with the *loudness* parameter. Using a value of 1.0 maintains the original amplitude of the *infile*, which will be heard as pretty much as the original, but with the superimpositions on top of it. *Loudness* is therefore a means of balancing the input and the processed sound components.

Musical Applications

This is a powerful and somewhat wild tool for producing distortion effects. The higher the value of *scaling*, the more the superimposed copies appear as a sheen of distortion above the original sound.

End of DISTORT FRACTAL

FRACTAL WAVE – Fractally distort an input sound or wavecyle

Usage

fractal wave 1 *inf outf shape* [**-m***maxfrac*] [**-t***str*] [**-i***warp*] [**-s**] [**-o**]
fractal wave 2 *inf outf shape dur* [**-m***maxfrac*] [**-t***str*] [**-i***warp*] [**-s**]

Modes

1 Fractally distort soundfile by transposition
2 Generate fractal wave from (mono) input wavecycle

Parameters

infile(Mode **1**) – input soundfile (1 or more channels)
or (Mode **2**) wavecycle (such as that generated by WAVEFORM
outfile – output soundfile
shape – breakpoint textfile of:

- (Mode **1**): time and transposition pairs, defining the contour of the largest fractal shape
  Time: 0 to file-length (secs); transposition: -12.0 to +12.0 (semitones)
- (Mode **2**) time and MIDI-pitch pairs, defining the contour of the largest fractal shape.
  Times start at zero and increase; final time indicates duration of pattern; MIDI: 0 to 127 (value at final time is ignored)

dur (Mode 2) – Output duration (Range: 1-7200 secs)
-m*maxfrac* – maximum degree of fractalisation (time-variable). If not set (or zero), fractalisation proceeds until the minimum possible wavelength is reached.
Range: 0-1000, but 10 is a safer maximum to avoid "too high MIDI pitch" error.
-t*str* – time stretch of fractal pattern (time-variable) Range: 1.0 to 2.0. If zero, no timestretching is done.
-i*warp* – interval warping of fractal pattern (time-variable). If set to zero, no warping is done.
-s – shrink pitch-intervals as fractal time-scales shrink
-o – breakpoint data read using time in outfile (Default: use time in infile).

Understanding the FRACTAL WAVE Process

In Mode 1, FRACTAL WAVE distorts the whole soundfile by adding transposed versions of *shape* data. In Mode 2, where the input is a wavecycle, it generates fractals using MIDI-pitch data (in *shape*) to define the pitch shape. Both versions of the shape are timed. Transposition happens over the total duration specified in *shape* and is then repeated over every resulting sub-unit of the pattern, over every sub-sub-unit, etc. until the smallest time-unit is reached.

The process may best be understood by taking a steady-pitched source and experimenting first with Mode 1, then extracting a wavecycle from the source using **WAVEFORM** and applying it to Mode 2.

In Mode 1, the output length is conditioned by the infile; in Mode 2, it is supplied by the user (*dur*). The fractal pattern may optionally be timestretched (*str*) and/or interval-warped (*warp*).

See also the spectral version **FRACTAL SPECTRUM**.

Musical Applications

...

End of FRACTAL WAVE

DISTORT HARMONIC – Harmonic distortion by superimposing 'harmonics' onto 'wavecycles'

Usage

distort harmonic *infile outfile harmonics-file* [**-p***pre_attenuation*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
harmonics-file – contains *harmonic_number amplitude* pairs

- The amplitude of the source sound is taken to be 1.0
- *Amplitude* range: 0.000031 to 32.0
- Harmonics range from 2 to 1024

-p*pre_attenuation* – apply attenuation to *infile* before processing

Understanding the DISTORT HARMONIC Process

Harmonic distortion multiplies and adds within a single 'wavecycle' – possibly several times. For each *harmonic_number* in the *harmonics-file*, DISTORT HARMONIC scales and copies the shape of the 'wavecycle' *harmonic_number* times and adds the result to the original at the given *amplitude* (relative to that of the *infile*). This is by direct analogy to harmonic additive synthesis, in which a complex pitched sound is created by adding sinusoidal partials. Indeed, DISTORT HARMONIC can be used for just this purpose by using a sine wave as input.

There is no internal scaling of harmonic amplitude values. It will be necessary in many cases to scale the *infile* with the *prescale* parameter to avoid overflow. *Prescale* is a multiplier, like an ordinary gain factor. (See the **Gain – dB Chart**).

Musical Applications

The higher 'harmonics' of the 'wavecycles' are heard as faster versions superimposed on the original 'wavecycle' and on the lower 'harmonics'. Therefore, the application is to add these higher and denser levels of distortion to the sound.

End of DISTORT HARMONIC

DISTORT INTERACT – Time-domain interaction of two sounds

Usage

distort interact mode *infile1 infile2 outfile*

Modes

1 Interleave 'wavecycles' from the two infiles
2 Impose 'wavecycle' lengths of 1st file on 'wavecycles' of 2nd

Parameters

infile1 – input soundfile number 1 (mono only)
infile2 – input soundfile number 2 (mono only)
outfile – output soundfile

Understanding the DISTORT INTERACT Process

In Mode **1** material from both soundfiles is audibly apparent due to the interleaving process. In Mode **2**, the distortion is almost total: the alteration of the 'wavecycle' lengths of the second sound changes it to a burbly, bubbly, seething mass.

Musical Applications

DISTORT INTERACT can be used to achieve distortion which combines data from two different sounds or distortion which totally alters a sound.

End of DISTORT INTERACT

DISTORT INTERPOLATE –Time-stretch file by repeating 'wavecycles' and interpolating between them

Usage

distort interpolate *infile outfile multiplier* [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
multiplier – (integer) number of times each 'wavecycle' repeats

multiplier may vary over time.

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT INTERPOLATE Process

With this process, the shape of a 'wavecycle' is transformed into that of the next over *multiplier* repetitions. Note that this is waveshape-based interpolation, not a spectral interpolation, and that the length of the 'wavecycle' is also transformed by the process.

The effect of the transformation is drastic, leading to a strongly granular *outfile*. The length of the *outfile* increases in step with the value of *multiplier*, as does the apparent pitchiness.

Musical Applications

The interpolation process adds a modulatory quality to the output, so that the successive wavecycles gliss and bend as they flow into one another. Even so, as *multiplier* increases, the perception of separate 'grains', i.e., 'wavecycles' increases. A value of 32, for example, changes the sound to a strange stream of modulating tones.

End of DISTORT INTERPOLATE

DISTORT MULTIPLY – Distortion by multiplying 'wavecycle' frequency

Usage

distort multiply *infile outfile N* [**-s**]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
N – multiplier (Range: 2 to 16, integer only)
-s – smoothing (try this if glitches appear)

Understanding the DISTORT MULTIPLY Process

The duration of the sound is not changed, only the frequency of the 'wavecycles', with the result that the pitch rises.

Musical Applications

The distortion is relatively mild, in that the original sound remains recognisable. However, the surface is textured and the pitch rises with each increase in the value of *N*. DISTORT MULTIPLY can be used, for example, to create high, modulating, grainy vocal sounds.

End of DISTORT MULTIPLY

DISTORT OMIT – Omit *A* out of every *B* 'wavecycles', replacing them with silence

Usage

distort omit *infile outfile A B*

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
A – number of 'wavecycles' to omit
B – size of group of 'wavecycles' out of which to omit *A* 'wavecycles'

A may vary over time, but must **always** be less than *B*.

Understanding the DISTORT OMIT Process

Because the omitted 'wavecyles' are replaced by silence, the overall duration of the sound does not change. The larger the proportion of 'wavecycles' omitted from *B*, of course, the more distorted the the sound becomes. This distortion is like a rough texturing, rather than the highly modulatory results of some of the other processes.

Musical Applications

This process can be used, therefore, to achieve a rough texturing with no loss of duration.

End of DISTORT OMIT

DISTORT OVERLOAD – Clip the signal with noise or a (possibly timevarying) waveform

Usage

distort overload 1 *infile outfile clip-level depth*
distort overload 2 *infile outfile clip-level depth freq*

Modes

1 Clip signal
2 Clip and add waveform

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
clip-level – level at which the signal is to be clipped (Range: 0 to 1) The signal level is renormalised after clipping.
depth – depth of the pattern of distortion imposed on clipped stretches of the signal. (Range: 0 to 1)
freq – frequency of the waveform imposed on clipped stretches of the signal

clip-level, *depth*, and *freq* may vary over time.

Understanding the DISTORT OVERLOAD Process

The *clip-level* parameter is rather like a 'gate' level. If the signal level is already high, anything over, for example, 0.1 is likely to push it into distortion, and values considerably higher than this will make it heavily distorted. However, if it only distorts, Trevor advises me, when it reaches a level of, for example, 0.99, it is not going to be distorted very often.

The sound doesn't actually have amplitude overload, because it is distorted by 'slicing off' the top (clipping) where it would have overloaded. The sound becomes loud and 'strained', like a voice which is shouting too loudly.

Mode **2** can add an extra ringing sound as the value for *freq* gets higher, e.g., 2000Hz and beyond.

Musical Applications

Given the trials made so far, this can be a fairly subtle effect, but the words 'straining', 'loud', 'uncompromising' seem appropriate as the amplitude gets pushed towards the top of the range.

End of DISTORT OVERLOAD

DISTORT PITCH – Pitchwarp 'wavecycles' of sound

Usage

distort pitch *infile outfile octvary* [**-c***cyclelen*] [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
octvary – maximum possible transposition up or down in (fractions of) octaves (Range > 0.0 to 8.0)

> Note that the pitch of **each** 'wavecycle' is varied by a random amount within the range of *octvary* octaves up to *octvary* octaves down: i.e., the value for *octvary* covers a total up/down range of 2 * *octvary*.

-c*cyclelen* – mamimum number of 'wavecycles' between the generation of transposition values (Range: > 1, Default: 64)

> *octvary* and *cyclelen* may vary over time.

-s*skipcycles* – (integer) number of 'wavecycles' to skip at start of file

Understanding the DISTORT PITCH Process

The random up/down movement of the 'wavecycles' within the total *octvary* range produces a great deal of bending of the sound, especially if the original alters its pitch a good deal. It is better, therefore, to start with relatively small values for *octvary* – e.g., less than 1 – so that you start to use this function with some degree of control over the results.

The full power of DISTORT PITCH doesn't really come into its own until time-varying parameters are used, especially for *cyclecnt*. Large values for the latter will serve to slow down the rate of change.

Musical Applications

DISTORT PITCH is useful for creating 'flexitones' (to coin a term) – with distortion, of course.

End of DISTORT PITCH

DISTORT PULSED – Imposed regular pulsations on a sound

Usage

distort pulsed 1 *infile outfile env stime dur frq frand trand arand transp tranrand* [**-s -e**]
OR
distort pulsed 2-3 *infile outfile env stime dur frq frand trand arand cycletime transp tranrand* [**-s -e**]

Modes

1 Impose impulse-train on source
2,3 Use a segment of the source as the looped content of a synthetic impulse-train

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
env – breakpoint envelope of impulse. This will be scaled to the duration needed.
stime – time in the source sound where the impulses begin. In Mode **3**, *stime* is given as **samplecnt**, i.e., number of samples
dur – length of time that the impulses continue
frq – number of impulses per second
frand – number of semitones by which to randomise the frequency of the impulses
trand – amount of time in seconds by which to randomise the relative time positions of amplitude peaks and troughs from impulse to impulse
arand – randomisation of the amplitude shape created by the peaks and troughs from impulse to impulse
cycletime – Mode **2**: duration in seconds of wavecycles to grab as sound substance inside the impulses
OR Mode **3**: number of wavecycles to grab as sound substance inside the impulses
transp – transposition contour of sound inside each impulse
tranrand – randomisation of transposition contour from impulse to impulse
-s – keep start of source sound, before impulses begin (if any)
-e – keep end of source sound, after impulses end (if any)

Process only works on MONO files

Understanding the DISTORT PULSED Process

Distort a sound by imposing a series of impulses on the source, or on a specific waveset segment of the source. An impulse is like a brief event created by a sharp envelope on the sound. The sound inside the impulse might glissando slightly, as if whatever is causing the impulsion has warped the sound by its impact.

End of DISTORT PULSED

QUIRK – Distort signal by raising sample values to a power

Usage

quirk quirk 1-2 *infile outfile powfac*

Modes

1 Apply power factor over amplitude range of individual half-wavesets
2 Apply power factor over amplitude range of entire signal

Parameters

infile – input soundfile (MONO).
outfile – output soundfile.
powfac – exponent. (Range: 0.01 to 100; <1 exaggerates signal contour; >1 smooths signal contour.)

Understanding the QUIRK Process

QUIRK introduces distortion into a signal by raising sample values to a power. The power factor (*powfac*) is applied to individual half-wavesets (pseudo half-cycles) or to the whole file. *powfac* values less than 1 tend to flatten the tops of the waveform, while values >1 tend to narrow the waveform shape, modifying the timbre accordingly. With higher values of *powfac* (e.g. >5), the signal is in danger of disappearing altogether; however, this is also a simple means of finding the times of its most prominent peaks.

Quirk may be compared with Cross-modulation (modify radical 6), which produces similar distortion if the signal is multiplied with itself.

Musical Applications

...

End of QUIRK

DISTORT REFORM – Modify the shape of 'wavecycles'

Usage

distort reform 1-7 *infile outfile*
distort reform 8 *infile outfile exaggeration*

Modes

1 Convert to fixed level square wave
2 Convert to square wave
3 Convert to fixed level triangular wave
4 Convert to triangular wave
5 Convert to inverted half-cycles
6 Convert to click stream
7 Convert to sinusoid
8 Exaggerate waveform contour

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
exaggeration – exaggeration factor (Range: 0.000002 to 40.0)

exaggeration may vary over time.

Understanding the DISTORT REFORM Process

This process reads each 'wavecycle' (sound inbetween zero crossings) and replaces it with a different waveform of the same length. Several waveform options are provided. Those which do not fix the amplitude level respond to the varying amplitude levels of each successive wavecycle, thus producing an additional (and arbitrary) distortion feature.

The 'fixed level' options produce consistently loud output.

The 'click' option replaces each 'wavecycle' with a mishmash of square pulses several samples long (random sizes), which sounds a bit like a rattle.

The 'sinusoid' option, as might be expected, is relatively smooth. It is actually a subtle form of filtering. The sine waves vary in length and amplitude because they are based on 'wavecycles' and because only some of the 'wavecycles' are replaced.

The 'exaggeration' option just seems to add a surface buzz.

Musical Applications

Modes **1** and **3** create quite vigorous forms of distortion. The other modes are more restrained, offering a variety of gently distorted versions of the original.

End of DISTORT REFORM

DISTORT REPEAT – Timestretch soundfile by repeating 'wavecycles'

Usage

distort repeat *infile outfile multiplier* [**-c***cyclecnt*] [**-s***skipcycles*]

Example command line:

```
distort repeat infile outfile 5 -c3 -s20
```

Parameters

infile – input soundfile to process (mono only)
outfile – soundfile output after processing
multiplier – number of times (integer) each 'wavecycle' (group) repeats
-c*cyclecnt* – number of 'wavecycles' (integer) in repeated groups
-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file

multiplier and *cyclecnt* may vary over time.

Understanding the DISTORT REPEAT Process

The repetition of the 'wavecycles' stretches out the sound, making it both longer and more granular in texture. This granularity is increased if (increasingly larger) groups of *cyclecnt* 'wavecycles' are used: then the whole group repeats *multiplier* times.

Musical Applications

DISTORT REPEAT produces long, grainy (distorted) sounds. The sense of stretching out the original is very apparent.

A significant application of DISTORT REPEAT is that, by increasing the *cyclecnt* factor, one crosses the pitch-perception boundary: that is, starting with a noisy sound in which all the wavecycles are randomly different, one ends up with, for example, 7 repetitions of the **same** wavecycle, followed by 7 of another and so on – and each of these comprise sufficient repetitions for us to hear **pitch**. Thus the noise source becomes a string of pitch beads, each of arbitrary timbre. With a *cyclecnt* of, for example, 128, one can even get a slowish random melody.

End of DISTORT REPEAT

DISTORT REPEAT2 – Repeat 'wavecycles' without time-stretching

Usage

distort repeat2 *infile outfile multiplier* [**-c***cyclecnt*] [**-s***skipcycles*]

Example command line:

```
distort repeat2 infile outfile 10 -c10 -s0
```

Parameters

infile – input soundfile to process (mono only)
outfile – soundfile output after processing
multiplier – number of times (integer) each 'wavecycle' (group) repeats
-c*cyclecnt* – number of 'wavecycles' (integer) in repeated groups
-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file

multiplier and *cyclecnt* may vary over time.

Understanding the DISTORT REPEAT2 Process

Repeating the 'wavecycles' without time-stretching (as in DISTORT REPEAT) enables you to increase the strength of the distortion with the *multiplier* parameter without making the output file any longer than the original. Larger values for *cyclecnt* increases the length of *infile* that is affected.

Musical Applications

Higher values for *multiplier* increase the distortion, while higher values for *cyclecnt* increase the length of *infile* that is processed as one unit.

Thus we could have:

- a low value for *multiplier* coupled with a high value for *cyclecnt* – this will produce a bit of distortion while the source remains recognisable
- a high value for *multiplier* coupled with a low value for *cyclecnt* – this will produces a great deal of distortion, but the effect is limited because only a few cycles are affected as a unit
- high values for both parameters – this appears to create the most distortion

End of DISTORT REPEAT2

DISTORT REPLACE – The strongest 'wavecycle' in a *cyclecnt* group replaces the others

Usage

distort replace *infile outfile cyclecnt* [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
cyclecnt – (integer) size of group of 'wavecycles'

cyclecnt may vary over time.

-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file

Understanding the DISTORT REPLACE Process

The replacing action serves to simplify the sound. Note that the single strong 'wavecycle' in the group will take the place of several others, which will be deleted. This simplification becomes extreme when the *cyclecnt* is high, leading to a 'sample-hold' kind of stepped effect. Time-varying *cyclecnt* makes it possible to introduce gradual change.

Musical Applications

With DISTORT REPLACE we can achieve a simplification of the sound, up to very clear 'sample-hold' type stepped tones.

End of DISTORT REPLACE

DISTORT REPLIM – Timestretch by repeating 'wavecycles' (below a specified frequency)

Usage

distort replim *infile outfile multiplier* [**-c***cyclecnt*] [**-s***skipcycles*] [**-f***hilim*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
multiplier – the number of times each wavecycle (group) repeats (Integer)
-c*cyclecnt* – the number of wavecycles in repeated groups
-s*skipcycles* – the number of wavecycles to skip at the beginning of the soundfile
-f*hilim* – the frequency below which cycles are counted

multiplier and *cyclecnt* may vary over time
NB: Works only on MONO soundfiles.

Understanding the DISTORT REPLIM Process

This function is like DISTORT REPEAT, but with a slight change. Here the length of wavecycle to be affected can be set. Thus, if you set a mid-range frequency, only those *below* that frequency will repeat, and the others (above the frequency) will be discarded (filtered out). Hence the name 'REP-LIM', meaning 'repeat (with a) limit'.

DISTORT REPLIM is therefore like a filtering program that also repeats wavecycles. In the DISTORT set, the wavecycles are wavelengths that occur between zero crossings, so distortion also occurs.

It is helpful to remember that wavelength is inversely proportional to frequency. Wavelength is the actual physical length of the oscillation, and frequency is the number of cycles that occur in one second (i.e., Hertz). These two aspects of sound are inversely proportional to one another: *P* = 1/*f*. For example, a sound oscillating at 100 Hz will have a period, i.e., a wavelength of 1/100 meters = 0.01 meters (0.39 inch). A sound oscillating at 1000Hz will have a wavelength of 1/1000 meters = 0.001 meters (0.039 inch).

Short wavecycles are therefore higher in pitch and long ones are lower in pitch. When the frequency setting for DISTORT REPLIM is high, the filter point is set higher and more of the sound will be retained. Here we are dealing with 'pseudo-wavecycles' (portions of soundfile between zero crossings), which is what introduces distortion into the equation.

Musical Applications

The net result of the function is to create repetition distortion while filtering out a user-definable amount of the higher frequencies. Remember that the relative amounts of high and low frequencies in the *infile* will affect the results.

End of DISTORT REPLIM

DISTORT REVERSE – Cycle-reversal distortion in which the 'wavecycles' are reversed in groups

Usage

distort reverse *infile outfile cyclecnt*

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
cyclecnt – number of 'wavecycles' in a reversed group (Range: > 0)

cyclecnt may vary over time.

Understanding the DISTORT REVERSE Process

Here the original soundfile is grouped into a series of 'wavecycles' with *cyclecnt* 'wavecycles' in each group. Then each of these groups of 'wavecycles' is reversed.

The term 'distortion' here is something of a misnomer, because no distortion process is applied to the 'wvecycles' themselves. Instead, *cyclecnt* sets the number of 'wavecycles' which are to be copied in reverse as a group to the *outfile*. For example, if *cyclecnt* = 3, 15 'wavecycles' reversed in groups of 3 will assume the order: 3-2-1, 6-5-4, 9-8-7, 12-11-10, 15-14-13. Thus, not only is the sound material backwards, but the reversed 1st 'wavecycle' is now adjacent to the reversed 6th 'wavecycle'.

This mimics the classical tape studio technique of cutting up a length of tape into segments (of varying lengths), reversing the segments, and joining up the reversed pieces.

The result will be similar to a random brassage because of the differing lengths of the 'wavecycles'.

The process moves steadily through the *infile* from beginning to end, so the normal order of the (reversed) events is preserved. It is surprising how normal the output can be. With mid-range values for *cyclecnt* (say, 30 to 100), one hears the original breaking up, but only with very large values for *cyclecnt* does one hear the sound sweeping backwards in large swathes. Again, it is a question of 'resolution': the size of the units being manipulated.

Musical Applications

A small value for *cyclecnt* will produce a grainy result, mid-values a 'broken up' result, and large values swathes of reversed sound. If the value for *cyclecnt* exceeds the number of 'wavecycles' in the *infile*, you will be told that the "sound source is too short...". **DISTORT CYCLECNT** returns the number of 'wavecycles' in a sound, should you want to provide a value for *cyclecnt* which is right up to the limit.

Reversing the output of DISTORT REVERSE turns the *cyclecnt* groups back the other way while reading the whole soundfile from back to front, producing an interesting mixture of forwards and backwards!

Using the time-varying option for *cyclecnt* provides an opportunity for dramatic or gradual changes in the output.

End of DISTORT REVERSE

SCRAMBLE – Scramble waveset order randomly or by size and level

Usage

```
scramble scramble   1-2  infile outfile dur seed [-ccnt] [-ttrns] [-aatten]
scramble scramble   3-4  infile outfile     seed [-ccnt] [-ttrns] [-aatten]
scramble scramble   5-8  infile outfile cuts seed [-ccnt] [-ttrns] [-aatten]
scramble scramble  9-10  infile outfile     seed [-ccnt] [-ttrns] [-aatten]
scramble scramble 11-14  infile outfile cuts seed [-ccnt] [-ttrns] [-aatten]
```

Modes

Reassembly is:

1 In random order.
2 In permuted random order (all wavesets used before any are re-used).
3 In order of increasing size (falling pitch).
4 In order of decreasing size (rising pitch).
5 In order of increasing size, in each segment.
6 In order of decreasing size, in each segment.
7 In order of increasing then decreasing size.
8 In order of decreasing then increasing size.
9 In order of increasing level.
10 In order of decreasing level.
11 In order of increasing level, in each segment.
12 In order of decreasing level, in each segment.
13 In order of increasing then decreasing level.
14 In order of decreasing then increasing level.

Parameters

infile – input soundfile (MONO).
outfile – output soundfile (MONO).
dur – duration of output file. (Range: 1 to 7200 secs)
cuts – textfile of (increasing) times in source: process in each separate segment. (Range: >0 to file-length)
seed – random seed; same seed with same random parameters gives same output. (Range: 0 to 256)
-c*cnt* – number of wavesets in waveset-groups to be scrambled. (Range: 1 to 256)
-t*trns* – range of any random transposition of wavesets. (Range: 0 to 12 semitones)
-a*atten* – range of any random attenuation of wavesets. (Range: 0 to 1)

Understanding the SCRAMBLE Process

As the name suggests, this process scrambles the order of wavesets (pseudo-wavecycles): randomly and by size and level. The fourteen modes set out the various possibilities. Some modes use a *cuts* datafile of times to define the segments. These can be set to the start of each segment that might be re-positioned. There is scope for optional random transposition and attenuation.

Musical Applications

...

End of SCRAMBLE

DISTORT SHUFFLE – Distortion by shuffling 'wavecycles'

Usage

distort shuffle *infile outfile domain-image* [**-c***cyclecnt*] [**-s***skipcycles*]

Parameters

infile – input soundfile (mono only)
outfile – output soundfile
domain – set of letters representing consecutive (groups of) 'wavecycles'
image – set of letters which forms some permutation of the *domain* set

- Items from *domain* may be reordered, omitted or duplicated.
- A typical *domain* could be **abcd**
- A typical *image* might be **aacccbdd** or **dccbba** or **dac** etc.
- The *domain* and *image* sets **must** be connected with a dash
- Full example: **abcd-aacccbdd**

-c*cyclecnt* – the size of 'wavecycle' groups to process: each character in *domain-image* represents *cyclecnt* groups of 'wavecycles' (Default: 1)

cyclecnt may vary over time.

-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file

Understanding the DISTORT SHUFFLE Process

A simple reordering in which *domain* and *image* have the same number of characters will suitably roughen up the sound. As the *image* duplicates characters, some time-stretching will occur.

Introducing higher *cyclecnt* values will mean that the *infile* is processed in larger units, increasing the recognisability of the original. Note that, in spite of the higher *cyclecnt*, time-stretching does not result – unless *domain* characters repeat in the *image*.

Thus, a *domain-image* of **abc-cba** would be:

3-2-1, 6-5-4 etc.

but with a *cyclecnt* of 5,

- **a** would comprise **1-2-3-4-5** (in the *domain*)
- **b** would comprise **6-7-8-9-10** (in the *domain*)
- **c** would comprise **11-12-13-14-15** (in the *domain*)

and the *image* **c-b-a** would now be:
11-12-13-14-15, 6-7-8-9-10, 1-2-3-4-5 – **26-27-28-29-30, 21-22-23-24-25, 16-17-18-19-20**

Note how the *cyclecnt* 'wavecycles' proceed sequentially forward in the file, even though the *image* involves a reversal of the *domain*. This is what produces the increased recognisability.

Musical Applications

The possibilities focus here on sculpting the roughness of the distortion along with time-stretch factors. Lots of room for playing with the *image* shapes.

End of DISTORT SHUFFLE

SPLINTER – Create splinters by repeating & shrinking selected waveset-group

Usage

splinter splinter 1-4 *infile outfile target wcnt shrcnt ocnt p1 p2*
[**-e***ecnt*] [**-s***scv*] [**-p***pcv*] [**-f***frq* |**-d***dur*] [**-r***rand*] [**-v***shrand*] [**-i**] [**-I**]

Modes

1, 3 Splinters lead into the original sound. Mode 1: splinters change pitch.
2, 4 Splinters emerge from the original sound. Mode 2: splinters change pitch.

Parameters

infile – input soundfile.
outfile – output soundfile
target – time in the source immediately before the desired waveset group. (Range: 0 to file-length secs.)
wcnt – number of wavesets used to create the splinter group. Must be < *frq* in 'length' (if *frq* used)
(Range: 1 to 256; maximum length 1 minute)
shrcnt – number of waveset-group repeats that are shrunk. (Range: 2 to 256)
ocnt – number of maximally-shrunken splinters beyond the shrink *shrct*. (Range: 0 to 256)
p1 – pulse-speed of waveset repetitions at the originating waveset (Range: 0 to 50 Hz).
If zero, Pulse-speed 1 is determined by the duration of the selected waveset-group (*wcnt*).
p2 – pulse-speed at end of repetitions (*shrcnt+ocnt* splinters away). (Range: 0 to 50 Hz)
If zero, Pulse-speed 2 = Pulse-speed 1.
-e*ecnt* – number of extra regular pulses beyond *shrcnt+ocnt*. (Range: 0 to 10000)
-s*scv* – shrinkage curve. (Range: 0.1-10; 1 = linear, >1 contracts more rapidly near originating waveset, <1 less rapidly)
-p*pcv* – pulse-speed curve. (Range: 0.1-10; 1 = linear, >1 accelerates more rapidly near originating waveset, <1 less rapidly)
-f*frq* (Modes1+2) – approx. frequency (1/wavelength) of maximally-shrunk splinters. (Range: 1000 Hz-Nyquist/2; default: c.6000 Hz)
-d*dur* (Modes3+4) – approx. duration of maximally-shrunk splinters (Range: 5 to 50 mS).
-r*rand* **–** randomisation of pulse timing; may vary over time. (Range: 0 to 1)
-v*shrand* – randomisation of pulse shrinkage; may vary over time. (Range: 0 to 1)
-i – mix all source into output. (Default: source used only where there are no splinters.)
-I – mix none of source into output.

Understanding the SPLINTER Process

SPLINTER cuts a short segment from the soundfile (at or near *target*) and repeats it as pulses, shrinking and optionally transposing it on each repetition. This pulsed portion is either followed by the reset of the soundfile (Modes 1,3) or precedes it (Modes 2,4). Alternatively, the option NOT to mix any of the source (flag **-I**) will just give the pulsed segment. In Modes 1 and 2, the splinters also change pitch.

The segment length is a group of pseudo-cycles (*wcnt*), so may be a bit hard to predict. *Shrcnt* and *ocnt* determine the number of pulses – those that are shrunk and those after the shrinkage, while *ecnt* can extend this number further. Pulse-speeds 1 (*p1*) and 2 (*p2*) set the speed of repetition, which can be arranged so that the repetitions speed up or slow down; if *p1* is zero, the speed is taken from the waveset group length *wcnt* . The speed of pulse and shrinkage can be skewed by *pcv* and *scv*, respectively.

Modes 1 and 2 gradually change pitch with each repetition; Modes 3 and 4 do not. *Frq* optionally sets a goal frequency to reach. In this context, the group of cycles has a notional 'frequency' (see also other waveset processes affecting frequency); if this exceeds *frq*, a CDP Error is raised.

Optional parameters include duration (*dur*, Modes 3 and 4), which sets the length of maximally shrunk splinters; *rand* and *shrand*, which randomize pulse and shrink timing, respectively.

Musical Applications

...

End of SPLINTER

DISTORT TELESCOPE – Time-contract sound by telescoping *N* wavecycles into 1

Usage

distort telescope *infile outfile cyclecnt* [**-s***skipcycles*] [**-a**]

Parameters

infile – input soundfile (Mono only)
outfile – output soundfile
cyclecnt – the number of 'wavecycles' in a group

cyclecnt may vary over time.

-s*skipcycles* – number of 'wavecyles' (integer) to skip at start of file
-a – telescope to an average 'wavecycle' length (Default: telescope to the longest 'wavecycle' length)

Understanding the DISTORT TELESCOPE Process

Although at first rather like DISTORT OMIT, here the 'wavecycles' are not deleted as such. Instead, they are superimposed (i.e., mixed) onto each other, with shorter 'wavecycles' being stretched to fit the longest one in each group of *cyclecnt* 'wavecycles'. The *outfile* will usually be much shorter than the *infile* and can be reduced to a mere blip with this process.

The **-a** flag tells the program to telescope to the **average** 'wavecycle' length, rather than to the longest. Since the longest 'wavecycle' in each group is compressed by this method, the *outfile* will be even shorter.

Musical Applications

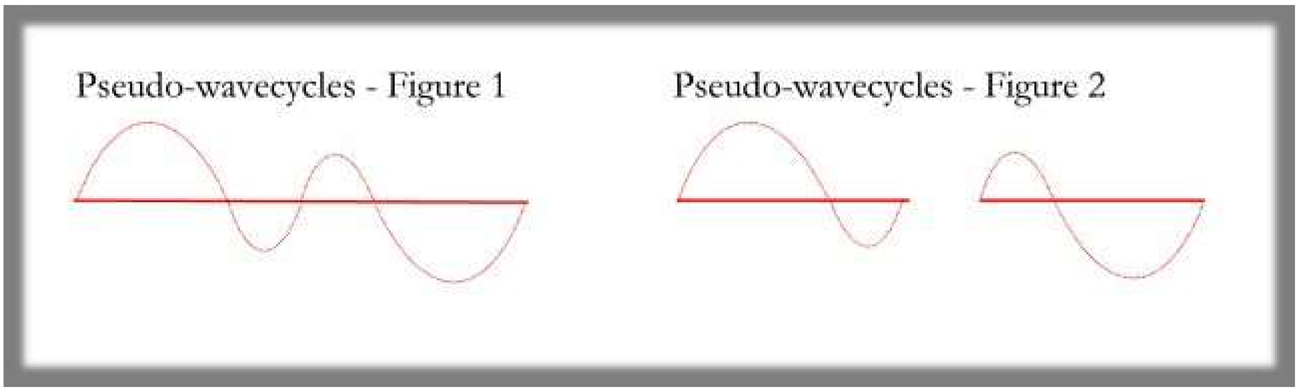
Interesting results can be achieved with small values for *cyclecnt*, the output tending to have a 'mushy' quality. It responds well to pitched material, producing a singing, if mushy, tone.

End of DISTORT TELESCOPE

Technical description of a 'Pseudo-wavecycle'

A pseudo-wavecyle is a way of dividing up the **whole** signal by taking chunks which stretch from **one** zero-crossing, through a **second** to a **third**. The reason these divisions of the signal are called 'pseudo' is because these segments of the waveform do not *necessarily* correspond to *real* wavecycles.

For example, a possible *true* wavecycle (corresponding to the fundamental of a pitch) of a sound might cross the zero more than twice, as in Figure 1. But this divides into two pseudo-wavecycles, as in Figure 2. Note that each of these pseudo-wavecycles crosses the zero at three points.



For more technical discussion of these processes, see Richard Dobson's
Introduction to the T Wishart DISTORT Functions

Introduction to the T Wishart 'Wavecycle' DISTORT Functions

by Richard Dobson

About the Wavecycle DISTORT Functions

Pseudo-wavecycles: a non-linear process

These functions have in common a graphical approach to sound transformation. That is to say, whereas orthodox time-domain transformation techniques apply some mathematical modification to successive samples (see, for example, ring modulation with **MODIFY RADICAL**), these DISTORT functions take as their basis the concept of the 'wavecycle', i.e., the set of samples between two sucessive zero crossings.

They are therefore highly non-linear, in the sense that the output of one of these DISTORT functions depends on the waveform contour of the infile. This contour does of course reflect the general sonic nature of the infile – a simple periodic sound will have regular, clearly defined and similar wavecycles, whereas a noisy or otherwise complex sound will be seen as a chain of wavecycles each different in shape, amplitude and duration.

The functions therefore take an essentially 'granular' view of a soundfile, much as a graphic artist might view an image as a set of variably-sized pixels (picture elements). Composers working with granulation techniques should find them especially interesting. While they can be applied to any sort of source sound, the most controlled and predictable results (given that they can often be impossible to predict!) are likely to be obtained from source sounds designed for the purpose of applying granular transformations.

For a simple geometric waveform such as a sine or triangle wave, the size of a wavecycle will indeed correspond to the true wavelength (or 'instantaneous pitch') of the input signal, but for more complex (i.e., wiggly) sounds, periodic or otherwise, it can be expected to vary widely and irregularly, as multiple zero-crossings within a single period of the infile are counted (erroneously, in a sense) as complete wavecycles in their own right. Therefore, although it is true to say that, the higher the pitch of the infile, the greater the number of wavecycles in each second, it is not completely true. Rather, there will be AT LEAST that many, and probably many more. In short, no general advice can be given about cyclecount parameter values, as these will depend almost entirely on the nature of the infile.

Predictability Factors

The factors determining the degree of predictability in using these functions may be summarized as follows:

- High predictability:
    - Simple geometric waveforms – sine, triangle, etc
    - Simple periodic waveforms – upper partials weaker than fundamental
    - Low rates of change in the above – simple vibrato or tremolo
    - Heavily low-pass filtered sounds
    - Output from **DISTORT OMIT**, **DISTORT REVERSE**, **DISTORT SHUFFLE** (see below)
    - Use of large wavecycle groups.
- Medium predictability:
    - Periodic waveforms with one or more partials stronger than the fundamental – many acoustic intrumental sounds
    - Moderately low-pass filtered sounds
    - Presence of transients – instrumental attacks, noise
    - Fast or irregular modulation
    - 'Granular' sounds
    - Low level of noise
    - Phase shifts
    - Use of medium-length wavecycle groups (brassage rates)
- High to impossible predictability:
    - Inharmonic sounds – bells, etc
    - High levels of noise
    - Rapid or chaotic transients
    - Speech
    - Dominant high frequencies
    - Processing confined to single wavecycles.

Phase Relationships

In a periodic (pitched) sound, the phase relationships between individual harmonic partials will often determine the number of zero-crossings internal to a period. Thus two soundfiles, identical apart from some (ofen inaudible) differences in phase, can lead to widely different outfiles when distorted by the same function. Note also that the wavecycle technique, while 'accurate' for simple pitched sounds, shows a deliberate disregard for phase or other forms of discontinuity, resulting in the introduction of various forms of mostly mild transient distortion (glitches) in the outfile. These may of course be desired, but in perhaps the majority of cases some form of filtering will need to be applied.

Given the graphic nature of these functions, it follows that inspection of outfiles and prospective infiles with a graphic display of the soundfile will help in understanding how a particular function works, and also in predicting the effect a function will have.

Experimental, with minimal error-checking

As is the nature of experimental software, error checking is kept to a minimum. Some essential tests are made, for example to avoid division by zero and similar hazards. Beyond these, few constraints are imposed on parameter ranges. It is especially important to remember the fact that you are working with digital samples - trying to envelope wavecycles in high frequency signals (where a wavecycle may only amount to a handful of samples) cannot be expected to produce predictable results. A 'robust' function might set a lower limit on the length of a wavecycle, but none of these functions do so – the only method is to use **DISTORT FILTER** (see below) to remove wavecycles below a user-defined cut-off length.

In almost all situations, therefore, putting prospective sounds through a strong low-pass filter such as **FILTER LOHI** will minimize unpleasant distortion and maximize interesting and attractive transformations. It follows that it is also worth using the highest possible sample rate.

Musical Applications

There are two primary uses for these functions. The first is simply to apply more or less extreme distortion or granulation to a specific infile. The second is the development of granular textures through the manipulation of wavecycles – infiles are thought of as 'libraries' or databases of wavecycles. It is important to develop a clear strategy for using a sequence of distort functions. A key stage in this strategy might well be step one: to create a soundfile of KNOWN WAVECYCLE CHARACTERISTICS.

Note that there are two general types of distort function: those that operate on **individual** wavecycles (**DISTORT DIVIDE**, **DISTORT FRACTAL**,**DISTORT HARMONIC**, **DISTORT INTERACT**), and the rest, which work on or output **groups** of wavecycles. **DISTORT MULTIPLY** also outputs groups of wavecycles when multiplying the wavecycle frrequency. Considerable flexibility can be obtained if the size of each group is highly composite (.e.g., a power of 2), so that 'groups within groups' can be manipulated. Note the use of a several linked DISTORT operations in the following procedures.

For example, weak wavecycles can be removed (**DISTORT OMIT**, **DISTORT DELETE**,**DISTORT REPLACE**). The remainder might be repeated with interpolation (**DISTORT INTERPOLATE**) to form larger groups, which are then reordered (**DISTORT SHUFFLE**) or enveloped (**DISTORT ENVEL**).

Alternatively, they might be enveloped first, and the resulting wavecycles reordered or repeated. A short section might be removed (**SFEDIT CUT**) and interpolated (**DISTORT INTERPOLATE**) over a long range, to create a timbrally simple file which might then be the source for a new sequence of operations, such as fractal or harmonic transformation (**DISTORT FRACTAL**, **DISTORT HARMONIC**).

Finally, it is well worth exploring the potential of the function **DISTORT PITCH** to colour vocal sounds. Voices can be made higher and lighter or deeper and more growley, either all at once or over a period of time.

Last updated: 2 May 2001

Last Updated 28 Apr 2023 for CDP8
Documentation: Archer Endrich, revised R.Fraser